# Lecture 9

# Ellipsoid II: Grötschel-Lovász-Schrijver theorems*

*I ♡ László Lovász.*

Ryan O'Donnell

We saw in the last lecture that the Ellipsoid Algorithm can solve the optimization problem

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b \end{aligned}$$

in time $\text{poly}(\langle A \rangle, \langle b \rangle, \langle c \rangle)$, where by $\langle z \rangle$ we mean the representation size of $z$. In proving this, we mostly disregarded numerical issues, such as how irrational numbers should be dealt with. In addition, we mostly stated the algorithm in terms of a general convex set $K$ rather than just a polytope, but then we neglected the whole host of issues surrounding general convex sets. In this lecture, we will fill in the remaining details. Largely these are either numerical or conceptual issues that require great length to be treated formally. As a result, this lecture will be relatively informal; a more precise and complete treatment is provided by Grötschel, Lovász, and Schrijver in [GLS88].

## 9.1 LP runtime

One loose end that we would like to mention quickly first is that it seems we should be able to do without the dependence on $\langle b \rangle$ and $\langle c \rangle$ in the Ellipsoid runtime. In some sense, the "real complexity" of the optimization problem should only depend on the polytope, not the direction of optimization. This intuition was proven correct by É. Tardos in [Tar86], in which she showed that the Ellipsoid Algorithm can solve linear programs in $\text{poly}(\langle A \rangle)$ time, via a reduction to linear programs of $\text{poly}(\langle A \rangle)$ size. One consequence of this is that if, for example, $A$ is a matrix whose entries are in $\{-1, 0, 1\}$, then solving the linear program can be done in strongly polynomial time. Matrices of this form do occur for many natural problems, max flow perhaps most prominent among them.

*Lecturer: Ryan O'Donnell. Scribe: John Wright.

## 9.2 Numerical details

We now treat the numerical issues that arise in the Ellipsoid Algorithm. First, we summarize the algorithm. Its task is to determine the feasibility (nonemptiness) of a convex set $K$ in $\mathbb{R}^n$. It receives two inputs:

1. A radius $R \in \mathbb{Q}$ such that $K \subseteq B(0, R)$.
2. A positive $\epsilon \in \mathbb{Q}$.

It outputs one of two things:

1. A point $s \in K$.
2. An ellipsoid $E \supseteq K$ such that $\text{vol}(E) \leq \epsilon$.

The running time of the Ellipsoid Algorithm is $\text{poly}(n, \langle R \rangle, \langle \epsilon \rangle, \text{“}\langle K \rangle\text{”})$, where by “$\langle K \rangle$” is meant something to do with the size of $K$. When we're solving linear programs, this is $\langle A \rangle + \langle b \rangle + \langle c \rangle$; for more general convex programs, we'll return to this issue.

The algorithm is iterative. At each step, it maintains an ellipsoid $E(s, Q)$ with center $s$ and matrix $Q$ which contains $K$. The algorithm begins each step by testing whether $s$ is in $K$. If so, then it outputs $s$. Otherwise, it (somehow) finds a vector $a$ such that $a^\top x < a^\top s$ for all $x \in K$. This is the “separating hyperplane”. Using this, it performs an update $E(s, Q) \to E(s', Q')$ to a new ellipsoid with center $s'$ and matrix $Q'$, and then it starts this process all over again. The update rule for the new ellipsoid is:

$$s' = s - \frac{1}{n+1} \cdot \frac{1}{\sqrt{a^\top Q a}} \cdot Qa \tag{9.1}$$

$$Q' = \frac{n^2}{n^2 - 1} \left( Q - \frac{2}{n+1} \cdot \frac{Qaa^\top Q}{a^\top Q a} \right)$$

One important thing to note is the square root in the denominator of Equation (9.1). We will return to this shortly. The point of this update rule is to produce a new ellipsoid of significantly smaller volume which contains the half ellipsoid formed by intersecting $E(s, Q)$ with the set $\{x : a^\top x < a^\top s\}$. By the choice of $a$ we know that this half ellipsoid contains $K$. The precise decrease in volume we proved in the last lecture was:

**Theorem 9.1.** $\text{vol}(E(s', Q'))/\text{vol}(E(s, Q)) \leq e^{-1/2n} \leq 1 - \frac{1}{3n}$.

Unfortunately, that square root means the possibility of irrational numbers, so that the Ellipsoid Algorithm couldn't update to $E(s', Q')$ even if it wanted to. To fix this, we'll need to show that if we perform approximate computations which are accurate to a sufficient degree, then everything comes out okay anyway. By necessity, this involves mucking around in some tedious numerical waters. To begin with, let $N$ be the total number of iterations, which we will pin down later. Then we'll modify the Ellipsoid Algorithm so that, when doing computations, it rounds all numbers to precision $2^{-p}$, where $p$ is set to $100N$. This solves the irrational number problem, but introduces a new one: the rounding changes the

center of the updated ellipsoid, and this will cause it not to contain the half ellipsoid that it was supposed to cover. To compensate for this, we adjust the algorithm so that it "blows up" each ellipsoid by a factor of $(1 + \frac{1}{10n^2})$ in each dimension. This is sufficient to make the ellipsoids completely contain the exact ellipsoids they are approximating, but it again introduces a new problem, which is that we can no longer guarantee as strong a bound as in Theorem 9.1. Hopefully, we can still get a strong enough bound, similar to Theorem 9.1 but with, say, a $1 - \frac{1}{10n}$ rather than a $1 - \frac{1}{3n}$, and if this is the case then we can just set the number of iterations $N$ to be something suitably large. In particular, taking

$$N := 10n\left(n\log(2R) + \log\left(\frac{1}{\epsilon}\right)\right) \tag{9.2}$$

is sufficient.

**Why it's okay.** We must ensure that the factor we have chosen to blow the ellipsoids up by does not blow them up too much. Doing this blow-up makes our approximating ellipsoid have $(1 + \frac{1}{10n^2})^n \approx (1 + \frac{1}{10n})$ times the volume of the exact ellipsoid. On the other hand, each exact ellipsoid has $(1 - \frac{1}{3n})$ times the volume of the previous ellipsoid. The net effect of these two opposing forces is that when we perform this rounding and blowing up, the updated ellipsoid has $(1 + \frac{1}{10n})(1 - \frac{1}{3n})$ times the volume of the original ellipsoid. This is no more than $(1 - \frac{1}{10n})$ times the volume of the original ellipsoid. The coefficient of 10 in Equation 9.2 is large enough so that the number of iterations yields a sufficiently small final ellipsoid.

There are some additional technical details that we need to attend to to ensure that everything still works after adding the rounding. For example,

**Lemma 9.2.** *In the exact version of the Ellipsoid Algorithm, let $B(s_k, Q_k)$ be the ellipsoid at step $k$, and let $\lambda_k$ be the minimum eigenvalue of $Q_k$. Then $|s_k|, \|Q_k\|, 1/\lambda_k \leq R \cdot 2^{2k}$, for all $k$.*

In particular, all of these bounds are at most $R \cdot 2^{2N}$. This is the key fact used to show that the rounding errors do not get magnified too much when being multiplied by matrices. Specifically, we just need $p$ to be large enough so that $2^{-p} \ll \frac{1}{R^2 \cdot 2^{2N}}$, and taking $p \approx \Theta(N)$ is sufficient.

## 9.3 Separation oracles

With that taken care of, we turn to applying the Ellipsoid Algorithm to more general convex sets. A difficulty that arises is that for a general convex set $K$, we may not have as convenient a description of it as we did in the case of sets of linear equations describing polytopes. Quickly glancing at the algorithm reveals that what we actually require from $K$ is quite limited: we should be able to test whether $s \in K$ (so-called "membership testing"), and if it isn't, we should be able to find a separating hyperplane. This minimal interface is formalized below in the definition of a separation oracle.

**Definition 9.3.** A *strong separation oracle* for $K$, when given as input $s \in \mathbb{Q}^n$, either returns "$s \in K$", or $a \in \mathbb{Q}^n$ such that $a^\top x < a^\top s$ for all $x \in K$.

There's one more tiny condition that we enforce on these oracles, and it is that the separating hyperplane it returns should have a manageable bit complexity—polynomial in the relevant parameters. Without this constraint, one could design a malicious separation oracle which always returns separating hyperplanes of exponential bit complexity, and then there's nothing the Ellipsoid Algorithm can do, no matter how clever we are in designing it.

We've already constructed a separation oracle, implicitly, for polytopes in the Ellipsoid Algorithm for linear programs last lecture. Let's go through an example for a different type of convex set. Let $K$ be the unit ball, i.e. $K = B(0, 1)$. Given $s \in \mathbb{Q}^n$, the separation oracle for $K$ is

- If $\|s\|^2 \leq 1$, return "$s \in K$". Otherwise, return $s$.

That $s$ is a good separating hyperplane is verified by the following simple calculation, which holds for $x \in K$:

$$s^\top x \leq \|s\| \cdot \|x\| \leq \|s\| < \|s\|^2 = s^\top s.$$

The first inequality is by Cauchy-Schwarz, the second is because $\|x\| \leq 1$, and the third is because $\|s\| > 1$.

Implementing the separation oracle is easy for the unit ball, but it may not be quite so easy for some less explicit convex set. Indeed, why would you want to run the Ellipsoid Algorithm on a general convex set? If you could compute the separation oracle, you might well be able to compute feasibility already. Well, we'd like to maximize a linear function over the set, the problem

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & x \in K. \end{aligned}$$

Recall that to solve this using a feasibility tester such as the Ellipsoid Algorithm, we do a binary search over the possible optimum values $\gamma$, at each step of the search testing feasibility of the region $K' := K \cap \{x : c^\top x \geq \gamma\}$. Say that we were given $\text{SEP}_K$, a separation oracle for $K$. We would somehow also need to design a separation oracle for $K'$. Fortunately, this quite easy. Given $s \in \mathbb{Q}^n$, the separation oracle for $K'$ is

- If $c^\top s < \gamma$, return $-c$. Otherwise, return $\text{SEP}_K(s)$.

## 9.4 General convex sets

This looks promising for optimizing a linear function over $K$, but when should we stop the binary search? With a linear program, we could stop in a fixed number of steps because we knew the optimal was a rational of bounded repesentation size. For a general $K$, on the other hand, it is possible, for instance, that the optimum is irrational, meaning that we have no hope of finding it. Examples of this aren't too hard to cook up. Consider

$$\begin{aligned} \max \quad & x + y \\ \text{s.t.} \quad & x^2 + 2y^2 \leq 1 \end{aligned}$$

The optimum is $\sqrt{3/2}$.

The issues aren't specific to the binary search either; the Ellipsoid Algorithm simply cannot test feasibility for general convex sets. The Ellipsoid Algorithm works by generating a series of smaller and smaller ellipsoids which contain $K$. In the case of linear programs, we were able to guarantee that either the associated polytope was infeasible, or it contained a ball of large enough radius. Thus, when the ellipsoids reach a certain small size, we can guarantee that their centers must be contained in the polytope, if it exists. But we cannot make a similar guarantee for a general convex set. In the extreme, the set may be a single point, in which case the Ellipsoid Algorithm can never hone in on $K$ completely.

Finally, even if the set $K$ starts out large, it is possible that the Ellipsoid Algorithm nonetheless has to determine feasibility of extremely small regions, due to the binary search. It could be that the convex set $K \cap \{c^\top x \geq \gamma\}$ is quite small. This is illustrated in Figure 9.1, where the binary search reduces the region the size of the feasible region.

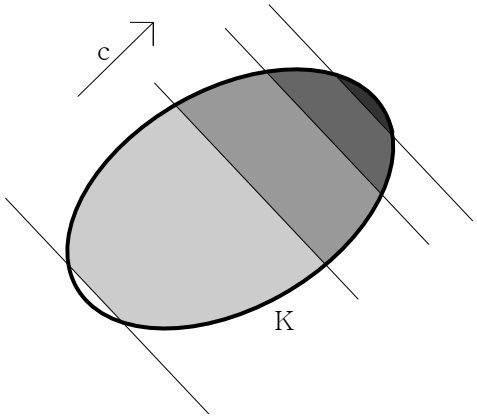


Figure 9.1: Binary search on the region $K$

At this point, we must either give up all hope of optimizing over general convex sets or define a relaxed notion of what it means to optimize (we'll opt for the latter). The relaxed notion of optimization involves demanding that the optimization only be approximate, in two separate ways. First, we saw that one of the two issues with optimizing over general $K$ is that $K$ may be too small. To rectify this, we introduce two approximate versions of $K$,

$$K^{+\epsilon} := \{y : \|y - x\| \leq \epsilon \quad \text{for some } x \in K\}$$

$$\text{and} \quad K^{-\epsilon} := \{x : B(x, \epsilon) \subseteq K\}.$$

Think of $K^{+\epsilon}$ as the set of points "almost in $K$", and $K^{-\epsilon}$ as the set of points "deep in $K$". Note that if $K$ is not full dimensional, then $K^{-\epsilon}$ is empty. In our relaxed form of approximation, we will only require the algorithm to optimize with respect to these sets.

The other area where we must allow for some approximation is in the objective value of the solution. Instead of requiring that it be optimal, we require only that it be near-optimal. With these notions in place, we can now state the following key definition:

**Definition 9.4.** The task of *weak optimization* of

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & x \in K. \end{aligned}$$

is, when given a positive $\epsilon \in \mathbb{Q}$, to

- Either output "$K^{-\epsilon}$" is empty;
- Or, find a $y \in K^{+\epsilon}$ such that $c^\top x \leq c^\top y + \epsilon$ for all $x \in K^{-\epsilon}$.

As stated before, we may never find a point exactly in $K$, or a point which optimizes exactly in $K$. So we must settle for weak optimization. One benefit of this is that we will only need a "weak" separation oracle to carry out weak optimization.

**Definition 9.5.** A *weak separation oracle* for K, when given as input $s \in \mathbb{Q}^n$ and a positive $\delta \in \mathbb{Q}$, asserts "$s \in K^{+\delta}$" if this is true, and otherwise returns $a \in \mathbb{Q}^n$ such that $\|a\|_\infty = 1$ and $a^\top x \leq a^\top s + \delta$ for all $x$ in $K^{-\delta}$.

The constraint on the infinity norm of $a$ is needed, because without it, setting $a := 0$ would always satisfy the approximate separating hyperplane constraint. Why can we now get away with using a weak separation oracle? Well, we're to have precision errors in the Ellipsoid Algorithm anyway, so a slight lack of precision here is ($\delta := 2^{-p}$) will not matter. With this in place, we are now ready to state the following general theorem:

**Theorem 9.6.** *Given $R$ and a weak separation oracle for a convex $K \subseteq B(0, R)$, we can weakly optimize over $K$ in time* $\text{poly}(n, \langle R \rangle, \langle \epsilon \rangle)$.

As a preview of things to come, we will use this to solve semidefinite programs in the next lecture.

## 9.5 Even more theorems

The remainder of this lecture will focus on a couple of the other results from [GLS88]. These are quite difficult to prove, so the discussion will be of a lighter nature.

### 9.5.1 Membership oracles

To start with, say you don't have a separation oracle for $K$, just a membership oracle. Recall that a membership oracle can test $s \in K$. This is a much more natural concept than a separation oracle, and it would be peachy if it were sufficient. Can we optimize over $K$, or even perform feasibility testing? Unfortunately, the answer is no. With only a membership oracle it's easy to see that an "adversarial set" $\tilde{K}$ can "evades" any points you query; you may never even "find" the set. However, say that in addition to a membership oracle, you are given a small ball that inhabits $K$: a point $s_0 \in \mathbb{Q}^n$ and a radius $r > 0$ such that $B(s_0, r) \subseteq K$. Then the following theorem says that this is sufficient to weakly optimize.

**Theorem 9.7.** *[YN76, GLS88] Given positive $R$, $r \in \mathbb{Q}$, a point $s_0 \in \mathbb{Q}^n$ such that $B(s_0, r) \subseteq K \subseteq B(s_0, R)$ and a weak membership oracle for $K$, then one can weakly optimize over $K$ in time* $\text{poly}(n, \langle R \rangle, \langle r \rangle, \langle \epsilon \rangle, \langle s_0 \rangle)$.

(Here is the definition of a weak membership oracle: when given $s \in \mathbb{Q}^n$ and a positive $\delta \in \mathbb{Q}$, it reports either that $s \in K^{+\delta}$ or $s \notin K^{-\delta}$.)

Ryan was not completely positive on how the proof of this goes, but here is what he thinks is the general outline. Basically, you want to use the weak membership oracle and the small ball of radius $r$ to implement some sort of separation oracle. Then you can appeal to Theorem 9.6. Remember that a separation oracle, given a point $s$, does a membership test and, if that fails, returns a separating hyperplane. We have a membership oracle, so the membership test is easy, but what to do for the separating hyperplane is a little more complicated.

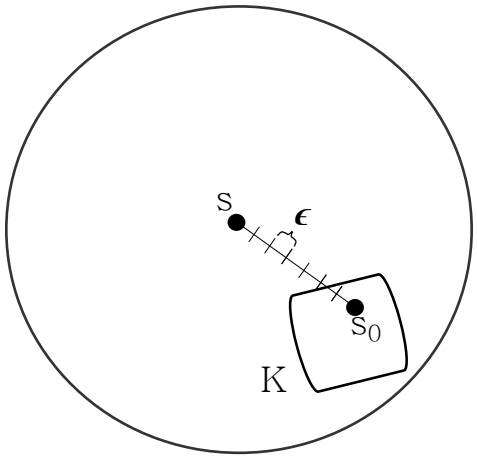


Figure 9.2: Crossing the boundary of $K$.

In this case, we know (roughly) that $s \notin K$, and we also that the point $s_0$ is in $K$. Thus, if we draw a line segment from $s$ to $s_0$, there is a point on the line segment where the line segment crosses the boundary of $K$. If we sample points on the line at intervals of roughly $1/\epsilon$ from each other, as in Figure 9.2, we can find "adjacent" points on the line which lie on either side of the boundary of $K$. Now, if we test a lot of directions in a small ball around these points, we can find what is roughly the the tangent to the boundary, and we can use this direction as the separating hyperplane. The main problem with this is that it runs in time $\text{poly}(1/\epsilon)$ rather than $\text{poly}(\langle \epsilon \rangle)$, but there is a variant of the Ellipsoid Algorithm ("Shallow Cut") for which this is sufficient.

### 9.5.2 General convex functions

A natural question is whether the functions we optimize must be linear, or whether they can be more general. Indeed, the results hold for any *convex* function $f : \mathbb{R}^n \to \mathbb{R}$. Recall that a convex function is one in which for all points $x, y \in \mathbb{R}^n$ and $t \in [0, 1]$,

$$f(tx + (1-t)y) \leq tf(x) + (1-t)f(y).$$

As always, we have to worry about what interface we have with $f$: an *oracle* for $f$, when given $x \in \mathbb{Q}^n$ and a positive $\delta \in \mathbb{Q}$, outputs a $y \in \mathbb{Q}$ such that $|f(x) - y| \leq \delta$. Furthermore, the oracle should run in time $\text{poly}(\langle x \rangle, \langle \delta \rangle)$. Given this, we have the following theorem:

**Theorem 9.8.** *Let $f$ be a convex function. Given positive $R$, $r \in \mathbb{Q}$, a point $s_0 \in \mathbb{Q}^n$ such that $B(s_0, r) \subseteq K \subseteq B(s_0, R)$ and a weak membership oracle for $K$, then one can weakly optimize $f$ over $K$ in time* $\text{poly}(n, \langle R \rangle, \langle r \rangle, \langle \epsilon \rangle, \langle s_0 \rangle)$.

We can even give a short "proof" of this (omitting a few easy-to-work-out details), by converting it into the case covered by Theorem 9.7. First, note that the region $L := \{(x, \gamma) \in \mathbb{R}^{n+1} : f(x) \leq \gamma\}$ is convex. Then weakly optimizing $f$ over $K$ is equivalent to weakly optimizing the following program:

$$\begin{aligned} \max \quad & 0 \cdot x + (-1) \cdot \gamma \\ \text{s.t.} \quad & x \in K, \quad (x, \gamma) \in L, \end{aligned}$$

which is just optimizing a linear constraint over a convex region.

### 9.5.3 Solving large LPs

For our last topic, we'll cover the well-known case of solving linear programs with exponentially many constraints (and only a separation oracle). Why is this any different from what we have already covered? The biggest problem is that the linear program may not be full dimensional. For example, in the following figure, we see centers $s_i$ drawing nearer to the region $K$ and the ellipsoids growing taller and flatter, but they never actually find $K$.

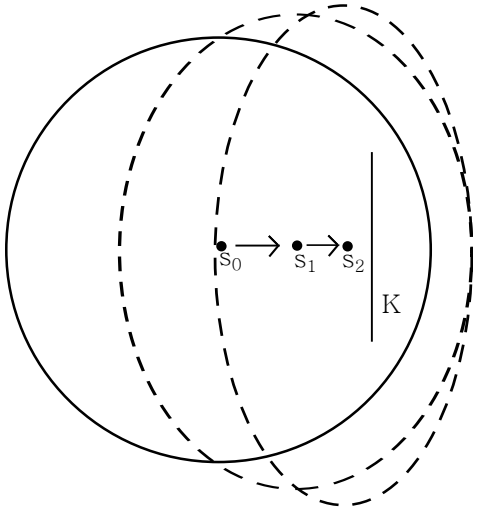


Figure 9.3: Ellipsoids closing in on a region $K$ not of full dimension.

This difficulty is quite inherent and cannot be easily evaded. How could one try to resolve it? Well, the ellipsoids are getting really flat, so they seem to identify a lower dimensional subspace. So if we could identify this subspace and "jump" into it, we could continue testing

for feasibility of $K$ in a subspace in which it is full-dimensional. This is indeed the idea behind the resolution of the problem. How to do this "subspace identification" is a difficult matter and it involves something called "simultaneous diophantine approximation". The scenario in simultaneous diophantine approximation is that there's a list of numbers, all almost expressible as integers over a certain bounded denominator, and the task is to find this denominator. This problem can be approximately solved by the so-called LLL algorithm, created by Lenstra, Lenstra, and Lovász. (The third major contribution in the area involving Lovász!)

The wonderful result of this is that we are able to get the following your-wildest-dreams-can-come-true theorem for large linear programs.

**Theorem 9.9.** *Let $K$ be $\{x : Ax \leq b\} \subset \mathbb{R}^n$, with each inequality $a^\top x \leq b$ satisfying $\langle a \rangle, \langle b \rangle \leq l$. (The number of inequalities here must be finite.) Assume access to a strong separation oracle for $K$. Then in time* $\mathrm{poly}(n, l)$*, you can:*

- *Perform feasibility/unboundedness testing.*
- *Perform exact optimization of primal and dual.*
- *Find an optimal vertex.*
- *etc.*

In essence, with this result, anything you can think of, you can do. The one downside is that in exchange for this, the proof is really hard.

**Wrap-up.** We've seen that the Ellipsoid Algorithm is capable of solving a diversity of convex optimization problems. In spite of its being relatively inefficient in practice, it has great theoretical value. Next lecture, we will apply the Ellipsoid Algorithm to solve semidefinite programs, an important class of optimization problems.

# Bibliography

[GLS88] Martin Grötschel, Lásló Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, 1988. 9, 9.5, 9.7

[Tar86] Eva Tardos. A strongly polynomial algorithm to solve combinatorial linear programs. *Operations Research*, 34(2):250–256, 1986. 9.1

[YN76] David Yudin and Arkadi Nemirovski. Informational complexity and effective methods of solution of convex extremal problems. *Economics and mathematical methods*, 12:357–369, 1976. 9.7